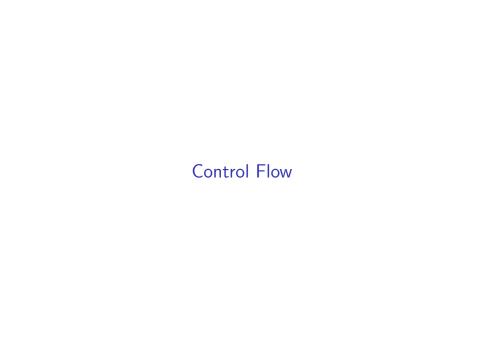
Stat 341 Lecture 3

Brad McNeney

2017-01-18

Control Flow

Reading from and writing to files



if and if-else

if tests a condition and executes code if the condition is true. Optionaly, can couple with an else to specify code to execute when condition is false.

```
if("cat" == "dog") {
  print("cat is dog")
} else {
  print("cat is not dog")
}
```

```
## [1] "cat is not dog"
```

for loops

```
Example:
n <- 10; nreps <- 100; x <- vector(mode="numeric",length=nreps)</pre>
for(i in 1:nreps) {
  # Code you want to repeat nreps times
  x[i] <- mean(rnorm(n))
summary(x)
      Min. 1st Qu. Median Mean 3rd Qu. Max.
##
## -0.70120 -0.23650 -0.02673 -0.01709 0.17750 0.97120
print(i)
```

```
## [1] 100
```

for loop index set

Index sets of the form 1:n are most common, but can be almost any atomic vector.

```
ind <- c("cat","dog","mouse")
for(i in ind) {
   print(paste("There is a",i,"in my house"))
}</pre>
```

```
## [1] "There is a cat in my house"
## [1] "There is a dog in my house"
## [1] "There is a mouse in my house"
```

while loops

Use a while loop when you want to continue until some logical condition is met

```
set.seed(1)
# Number of coin tosses until first success (geometric distn)
p <- 0.1; counter <- 0; success <- FALSE
while(!success) {
  success <- as.logical(rbinom(n=1,size=1,prob=p))
  counter <- counter + 1
}
counter</pre>
```

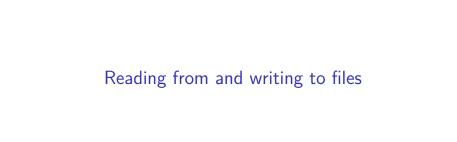
```
## [1] 4
```

break

▶ break can be used to break out of a for or while loop.

```
for(i in 1:100) {
  if(i>3) break
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```



Native format

- ▶ Use save() to save R objects to an "R Data" file.
 - save.image() is short-hand to save all objects in the workspace

```
x <- rnorm(100); y <- list(a=1,x=x)
save(x,y,file="test.RData") # Or .rda, or ...</pre>
```

Load R Data files into the workspace with load().

```
load("test.RData")
file.remove("test.RData")
```

```
## [1] TRUE
```

Table format files

- read.table() is the main function for reading tabular data from plain-text files.
 - read.csv() and read.delim() are basically read.table() with defaults for reading comma- and tab- delimited files.
- write.table(), write.csv() and write.delim() are the analogous functions for writing tabular data

```
write.table(matrix(1:9,3,3),file="test.txt")
test <- read.table("test.txt")
file.remove("test.txt")</pre>
```

```
## [1] TRUE
```

test

```
## V1 V2 V3
## 1 1 4 7
## 2 2 5 8
## 3 3 6 9
```

Reading files from a URL

▶ load(), read.table(), etc. can read data from a URL.

```
baseURL <- "http://people.stat.sfu.ca/~mcneney/Teaching/Stat341/
rdURL <- url(paste0(baseURL,"Data/PorschePrice.rda"))
load(rdURL)
head(PorschePrice)</pre>
```

```
## Price Age Mileage
## 1 69.4 3 21.5
## 2 56.9 3 43.0
## 3 49.9 2 19.9
## 4 47.4 4 36.0
## 5 42.9 4 44.0
## 6 36.9 6 49.8
```

```
csvURL <- url(paste0(baseURL,"Data/PorschePrice.csv"))
PorschePrice <- read.csv(csvURL)</pre>
```

Reading more complex text files

##

- ▶ Defaults for read.table() are not always what you want.
 - ▶ In particular, the default for reading columns that include text is to coerce to a factor.
 - Also replaces spaces in column headers with ...

```
exURL <- url(paste0(baseURL, "Data/Ex1_1_4.txt"))
ex <- read.table(exURL,header=TRUE,sep="\t")
# same as ex <- read.delim(exURL)
ex
##
    ID Initials Date.of.purchase amount
## 1 3
           SEKK 10/23/1995 $5.00
## 2 1 AGKE 08/03/1999 $10.49
           SBKE 12/18/2002 $11.00
## 3 2
str(ex)
  'data.frame': 3 obs. of 4 variables:
##
   $ ID
                    : int 3 1 2
   $ Initials
                    : Factor w/ 3 levels "AGKE"."SBKE"...: 3 1 2
##
##
   $ Date.of.purchase: Factor w/ 3 levels "08/03/1999", "10/23/1995",...: 2 1 3
   $ amount
                     : Factor w/ 3 levels "$10.49", "$11.00", ...: 3 1 2
```

stringsAsFactors

- Reading columns that include characters in as factors is controlled by a global option in your R session called stringsAsFactors, set to TRUE by default.
- ▶ If you want to set to FALSE for an R session type options(stringsAsFactors = FALSE) into the Console.
- ▶ An alternative is to over-ride the default in the call to read.table():

Post-processing: dates

[1] 1380 1233

Date.of.purchase should be coerced to a Date object.

```
ex2$Date.of.purchase <-
 as.Date(ex2$Date.of.purchase, "%m/%d/%Y")
str(ex2)
## 'data.frame': 3 obs. of 4 variables:
##
   $ TD
                     : int 3 1 2
## $ Initials : chr "SEKK" "AGKE" "SBKE"
##
   $ Date.of.purchase: Date, format: "1995-10-23" "1999-08-03"
##
   $ amount
                     : chr "$5.00" "$10.49" "$11.00"
diff(ex2$Date.of.purchase)
## Time differences in days
```

Post-processing: strings

[1] "\$5"

- ▶ Will probably want to remove the \$ in amount and coerce to numeric.
- Many options for manipulating strings.

"\$10.49" "\$11"

► For now, just mention substr() and paste() (more on string manipulation in Chapter 2):

```
maxStringLen <- 10 # allows for amounts up to $999999.99
ex2$amount <- as.numeric(
   substr(ex2$amount,start=2,stop=maxStringLen)
   )
str(ex2)</pre>
```

Reading and writing Excel files

- ▶ If you have a working copy of Excel, the easiest approach is to work with .csv format and use read.csv() and write.csv().
- However, there are functions in several R packages for reading directly from an Excel file.
 - ► See, for example, read.xls() from the gdata package.